	Professorship Computer Engineering **Automotive Software Engineering Practical** Prof. Dr. Wolfram Hardt, Dipl.-Inf. Norbert Englisch, M.Sc. Owes Khan, M.Sc. Felix Hänchen	TECHNISCHE UNIVERSITÄT CHEMNITZ
SS 2017	Automotive Software Engineering Practical Course - Unit 1 Basics of automotive software engineering and an introduction to the evaluation boards	06.04.2017

Content

1. Basics

For more than a hundred years, motor vehicles are produced and improved. The use of microelectronics provided new possibilities for engineering and implementation of complex functionalities for automobiles. Today most car drivers use their automobile for more than one hour per day, this results in increasing demands of the driver to the vehicle. To meet all the demands the usage of hardware and software in automobiles increases rapidly. The fast and reliable reaction of automotive functions is assured by software, electronic control units and their communication among each other. Furthermore former mechanical functions were replaced by software, electronic control units, sensors and actuators. Today luxury class automobiles contain up to 80 connected electronic control units i.e. engine management, airbag control, power locks and electric windows. In the case of the electric windows the software automatically reverses/stops the window motor if it senses an obstruction while closing.

Nowadays 90% of the innovations in car development concern the electronics and the software. That includes the conversion of classic mechanic functions as well as new features for example the automatic parking system.

For the practical automotive specific terms are necessary which are explained in the following table:

Term	Definition
Embedded system	Computer with special purpose. Serves to control and monitor a system.
Distributed systems	Computer network which communicates by message exchange.
Interactive systems	Hardware and software components which process user input and return results.
Reactive systems	Hardware- and software components which interact permanently/cyclic with the environment. Reactive systems normally do not terminate.
Real-time system	Computer system, which guarantees the calculating of a job within a predefined deadline (time limit). It is differentiated between hard, firm and soft real time.
Communication	The communication in the automotive domain describes the message exchange of distributed and embedded systems with the help of protocols.
Reliability	Ability of a system to consistently perform its required function within a specified time limit without degradation or failure.
Availability	Proportion of time a system is in a functioning condition.
Security	Extent to which a system is protected against data corruption, destruction, interception, loss or unauthorized access.
Monitoring	Observing of the current system status to detect failures and to initiate counteractions.
Electronic control unit	An electronic control unit (ECU) is an embedded system which controls one or more of the electrical subsystems in a vehicle. Sensors, actuators and ECUs are often one unit in mechatronic systems.
Sensors	Electronic device used to measure a physical quantity such as temperature, pressure or loudness and convert it into an electronic signal
Actuators	Transformation of electronic signals from an ECU into mechanical respectively physical quantities.

Table 1: Automotive specific terms and definitions

1.1. ECUs in Modern Automobiles

Increasing customer demands, such as lower fuel consumption, improving of driving safety and comfort are closely related to the use of more and more electronics and software in modern automobiles. The requirements of the ECUs in today´s automobiles are the following:

- Reliable under rough and changing ambient conditions like temperature, humidity and vibration
- High demands to the dependability and availability
- High demands to the security

ECUs work as a reactive system. The driver has no influence on the functionality except from activating sensors (e.g. switches).
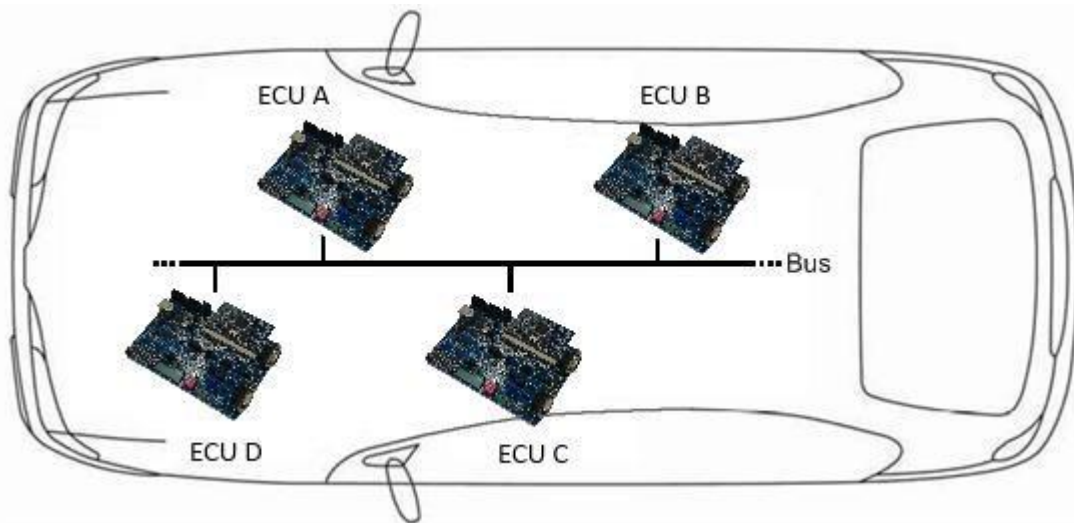

Figure 1: ECUs connected by a bus system cf. [1]

This easy, general example demonstrates the interconnection of the controllers and their communication over a bus. It is typical for a bus that the data transfer is realized via messages. Therefore the developer has the ability to send information resp. messages (for example the outdoor temperature) from one ECU to 1 or several others over the bus. The installed ECUs are divided into the following subsystems:

- Engine
- Chassis
- Car body
- Multimedia
- Security

To stress the importance of the interconnection and the categorization of ECUs, the following example shall be used. The ACC (Adaptive Cruise Control) is an add-on for the cruise control. The system maintains a steady speed as set by the driver and automatically keeps the necessary securitydistance. When approaching a slower vehicle the speed is adjusted to the speed of the preceding vehicle and thus the security distance is kept. If the distance increases the systems accelerates up to the defined speed. To realize this function several ECUs are necessary in the automobile. The ACCECU communicates with the engine-, the ESP- and the gearbox ECUs to adjust the driving speed. ESP is necessary to selective decelerate individual wheels to avoid the vehicle breaks away.

This simple example shows the importance of the communication between the affected ECUs. Only by interaction of the different ECUs the functionality of the ACC can be displayed. It also shows that simple features need complex implementations.

1.2. Sensors – ECUs – Actuators

Sensors deliver the data which is then analyzed by the ECU. They measure physical quantities and convert them into electric quantities, so they can be processed. There are various types of sensors that

react to different physical quantities, such as air pressure, humidity, light intensity, distance, temperature or density. The ECU periodically reads the sensor data and processes the implemented software. At this point it is decided whether an actuator is activated or not. If the sensor data differs between the periods in most cases the actuators are accessed by the ECU to react on the environmental changes. Furthermore the actuators can be accessed if sensor data have arrived at certain threshold. A simple example is the headlight which is automatically switched on at nightfall. A photodiode measures the light intensity and forwards it to the ECU. If the intensity of the light falls below a threshold the headlights are turned on.

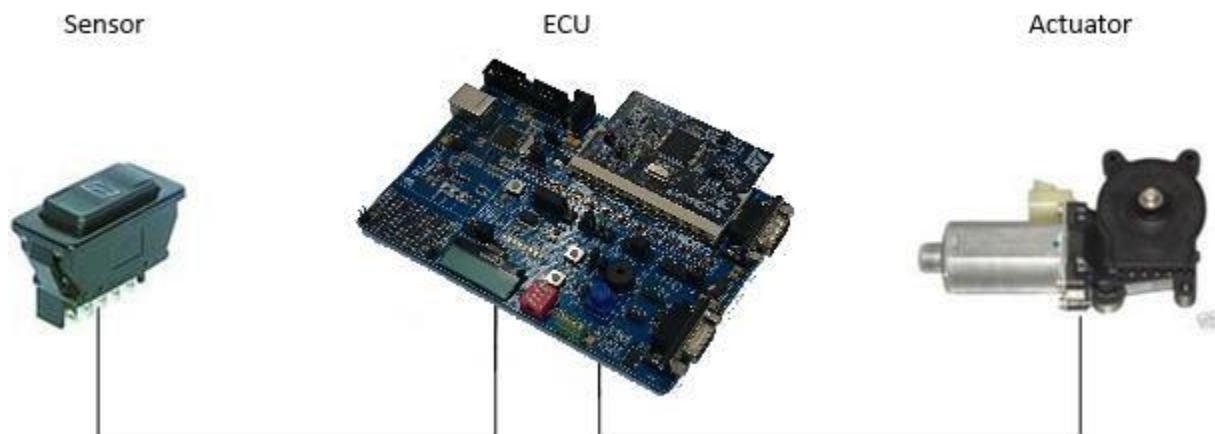The following graphic shows the relation between sensor, ECU and actuator.

**Figure 2: Structure of an electronic power window**

The graphic shows the function of an electronic window. A simple switch serves as the sensor. Once the switch is pressed the ECU activates the actuator, in this case the electric window motor. The ECU is responsible for automatic reversing the electric windows if it senses an obstruction while closing.

1.3. Evaluation Board

The evaluation board from Raisonance serves as development and testing platform for the SPC560P controller from STMicroelectronics. It can be used as a standalone application or in a network. Furthermore it is possible to flash developed software onto the board and to run it. If software is transferred to the board, it can also be operated with a power supply and without a computer. The Board provides different interfaces to access external devices, like sensors and actuators. The following graphic shows the provided sensors, actuators und further interfaces on the board:

Digital Outputs Area

- Eight red transistor-driven LEDs (D0 – D7) that light up when the command is low. Current is supplied by the power supply (9V or USB).
- Eight jumpers enabling the independent use of each LED. The use of an LED is enabled when the corresponding jumper is plugged in.
- LCD
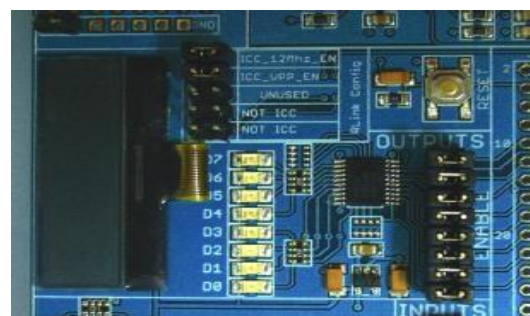
Digital Inputs Area

- Two-position jumper to choose the polarity of the BT6 push button.
- Two push buttons and four switches. When the switches are on or the buttons pressed, the corresponding signal is tied to the ground (except for BT6 which depends on the chosen polarity).

Analog Area

- Four-position analog connector which includes:
- Two analog inputs that can be connected directly to the ADC input pins of the microcontroller
- Analog output resulting from the integration of a PWM output. The integration is done with a simple RC filter with a characteristic time of 100ms
- Ground connection
- Potentiometer, that can be connected directly to the ADC input pins of the microcontroller
- Temperature sensor, that can be connected directly to the ADC input pins of the microcontroller. The output voltage of this sensor is given by the formula $V=0.01(T+1)$.
- Buzzer that can be connected to the PWM microcontroller output through the PWM/BUZ jumper

Communication Area

- Three-position connector for I²C communication
- RS-232 connector,
- Four-position connector for SPI communication

Secondary Serial / Can Area

This area allows the use of other serial communication protocols (CAN for example).

There are several sensors and actuators already on the board, no more components are needed for simple experiments. Please study the tasks under point 2 and the source code files in the appendix before the experiment.

1.4. Practical Examples

Development of functionality for the Yellow Car

The Yellow Car provides many sensors and actuators which can be used for the implementation of different functions. Amongst others the flashing indicator control and engine management are implemented.

Interaction of sensors and actuators via an ECU

The window regulator is a practical example which shows the interaction of a sensor, an ECU and an actuator. The switch panel serves as the sensor and the window motor as the actuator. The entrapment protection and the up and down moving functions are a software implementation on the ECU.
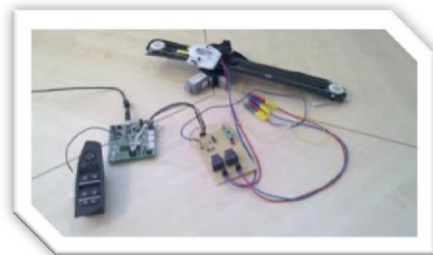
Communication between ECUs

To enable the communication between ECUs, messages are defined and transmitted over the bus. One ECU serves as the user interface (e.g. the steering wheel) which sends messages to the indicator ECU when an indicator button is pressed. There the message is then analyzed and afterwards the respective indicator light is turned on/off.
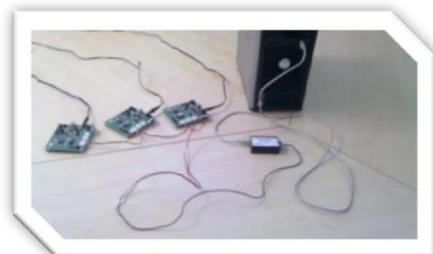
Instrument Cluster of a BMW 7 Series

The armature serves to demonstrate a cockpit. The infotainment-system works with the help of an evaluation-board

2. Experimental Procedure

2.1. Programming Microcontrollers

Microcontrollers are integrated circuits which contain a processor, memory and specific hardware periphery modules. The hardware modules can perform digital and analog functions like general purpose in- and output (GPIO), pulse width modulation (PWM), timers, and analog to digital converters (ADC). Sometimes peripheral modules for communication are also integrated, e.g. CAN, FlexRay, Ethernet, SENT, I2C, and SPI. These modules can be accessed and configured by integrated registers.

Microcontrollers are generally programmed in Assembler or in C language. In this practical course, the higher level language C is used to program the ECUs. This kind of programming is similar to programming software for PCs. But in embedded systems, the programs access the registers directly to configure the peripheral modules by setting certain bits to activate a desired peripheral function on a particular microcontroller pin. Once the pins are configured for a particular peripheral function, other registers are used either to control or to receive a value from the environment. For this purpose the memory addresses of the registers are used, which are defined in the reference manual of the specific microcontroller. Hence we use register addresses rigorously to access information from peripheral modules.

To ease this process, header files with macros are often delivered by the manufacturer of the microcontroller. In this practical course macros are defined in the "jdp.h" header file, which can be found in the project folder.

In general, the pins of the microcontroller are organized by ports. Every port of the microcontroller has a set of pins. Each port can only perform certain tasks due to limited support for remapping peripheral functions. More details of it can be found in the reference manual.

Figure 3 shows an extract of the reference manual of the microcontroller, which can be found at [2]. It shows the structure of the pad configuration register (PCR) of the System Integration Unit Lite (SIUL) module, which is used to configure digital in- and outputs.

Figure 3: SIUL pad configuration register (PCR) [2]

To access the PCR registers in C code, it is possible to directly access the base address of the registers or to write **"SIU.PCR[X].R"**, which is a macro defined in the "jdp.h" header file. The meaning of bit fields is also described in the reference manual.

Example:

Objective: Configure port **B8** of an SPC560L5 microcontroller as a digital output and then set it low.

Look into the reference manual and find the port pin. As shown in Figure 4, port pin **B8** supports general purpose in- and output (**GPIO**) and the corresponding PCR number is 24 (**PCR[24]**).

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	100-pin	144-pin
B[7]	PCR[23]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[23] — — — AN[0] RXD	SIU Lite — — — ADC0 LIN0	Input Only	—	—	29	43
B[8]	PCR[24]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[24] — — — AN[1] ETC[5]	SIU Lite — — — ADC0 eTimer0	Input Only	—	—	31	47
B[9]	PCR[25]	ALT0 ALT1 ALT2 ALT3 —	GPIO[25] — — — AN[11]	SIU Lite — — — ADC0 – ADC1	Input Only	—	—	35	52

Figure 4: Pin list and multiplexing [2]

To configure an I/O pin the "output buffer enable" bit (**OBE**) must be set. In Figure 3 the OBE bit is at position 6. To set this bit, "0x0200" (hexadecimal), "0b0000001000000000" (binary), or "512" (decimal) could be written to the register, but there are also several other ways to set this bit. The compiler used in this practical course doesn't care about the way it is done. In C it can look like this for port **B8** of the STM SPC560L5 microcontroller:

```
SIU.PCR[24].R = 0x0200;  /* configures port B8 as output */
```

To activate the output as "low signal", a "0" must be written to the corresponding general purpose digital output register (**GPDO**).

```
SIU.GPDO[24].R = 0;       /* sets port B8 low           */
```

Every module on the microcontroller can be configured in a similar way. In the reference manual all functionalities and registers are described. These are the basics of microcontroller and driver programming.

Note: Every microcontroller has its own individual modules and registers. That's why it is necessary to read the reference manual and to program new drivers when the hardware is changed.

2.2. Initial Setup

To implement functions on the ECU, the *"Raisonance Ride7"* IDE will be used. To flash code to the board *"Rflasher7"* is used.

The compiler must be started by opening *"WindRiver -> Development Shell"* in the start menu or on the desktop.

Then it is necessary to enter the command *"ride7"* in the development shell to start the IDE.

It is mandatory to do this first. Otherwise the code will not be compiled. The flashing program can be found under *"Raisonance tools -> Ride7 -> RFlasher7"* in the start menu. The project *"Unit1\Unit1"* (Source code in the appendix) has to be opened in the ride7 IDE. Analyze and study the source code on the file *"main.c"* and the user interface of Ride7 and Rflasher7. To flash the program, connect the board with the computer, load the output file *"Unit1\output\unit.s19"* and click the *"Go"* button. Please note that the output file has to be created by the compiler, hence pressing the *"Build"* button from the IDE will create (if there are no errors in the code) an output file in the specified location.

2.3.   Task 1

The I/O pins of the main controller situated on the daughter board are multiplexed (they can be used for multiple functions). The **aim** of the first task **is to configure these pins as general purpose output for the LEDs**. Therefore it is necessary to refer to the additional document *("pin_muxing.pdf")* which is provided. In this document the corresponding *"PCR"* register associated with each port pin can be found. The first page of the additional document shows the mapping of peripherals onto different pins of the controller. The correct pins/ports for the LEDs (*LED0 - LED7*) with the corresponding *"PCR"* register number can be found in the additional document.

The table below (Table 2) shows the IO port pin mapping of the used evaluation boards. As an example *LED6* corresponds to port pin *PD11*, and the *"PCR"* register number for this LED from the additional document is *"59"*. To use an LED the corresponding pin has to be configured as output. Furthermore, since the pins are multiplexed, the correct function of the pin has to be chosen. For the LEDs this can be done by writing *"0x0200"* to the corresponding configuration register (*"PCR"*).

The function *"SIU_Init()"* in file *"SIU.c"* shall be used for this task.

In order to light an LED, a low signal should be applied on the I/O pin. This is possible by writing *"0"* in the corresponding output register (*"GPDO"*). For turning off a high signal should be applied by writing *"1"* to the corresponding register.

Example for lighting an LED: *"SIU.GPDO[X].R = 0;"* where X stands for the corresponding *"PCR"* register.

Feature	CPU Pin Name	Comments
LED7	PA11	Light when command is low

LED6	PD11	Light when command is low
LED5	PD13	Light when command is low
LED4	PD14	Light when command is low
LED3	PA12	Light when command is low
LED2	PA13	Light when command is low
LED1	PC10	Light when command is low
LED0	PA9	Light when command is low
BT6	PA0	No pull-up/down, off=float, on=low/high jumper
BT5	PA1	No pull-up/down, off=float, on=low
SW4	PA2	No pull-up/down, off=float, on=low
SW3	PA3	No pull-up/down, off=float, on=low
SW2	PA4	No pull-up/down, off=float, on=low
SW1	PC12	No pull-up/down, off=float, on=low
ANA IN1	PC1	External analog input
ANA IN2	PC2	External analog input
ANA OUT	PC9	PWM connectable to the buzzer or ANA OUT
POT	PE1	Potentiometer analog input, min=0V, max=VDD
TEMP	PE2	Temperature sensor analog input

Table 2: IO Port Pin Mapping cf. [3]

2.4. Task 2

The light sensor (photoconductive cell) is used in automobiles to automatically turn on/off the head lights. The implementation of this exercise shows the automatic activation of actuators regarding sensor data.

Subtask 1:

The value of the sensor, which is converted by the ADC module of the controller, is stored in the register *"ADC_0.CDR[2].B.CDATA"*.

This subtask concerns with displaying the value of the light sensor data through LEDs.

The **controller pin**, which is mapped to the ADC module, **has to be configured for analog input** which can be done by writing *"0x2500"* to the corresponding *"PCR"* register. Please refer to Table 2 for the corresponding port/pin. The light sensor is connected to the "*ANA IN1"* input pin. The code for the analog pin configuration should be entered in the *"SIU_Init()"* function.

The function *"showData(value)"* contains the code for the different value levels. **The task is to glow LEDs corresponding to each level**, i.e. for the highest value of the sensor all LEDs should be switched on. Please check and modify the function *"showData(value)"* in the provided code.

Subtask 2:

This subtask concerns with the **usage of the potentiometer** provided on the board. The converted value from the potentiometer is being stored in the register *"ADC_0.CDR[4].B.CDATA"*. Use this data register, instead of light sensor data register, with the same function (*"showData(value)"*).

Please note: To read converted values from the ADC data register, the corresponding controller pin has to be configured in the same manner as the light sensor. The corresponding *"PCR"* register is not the same in this case!

Subtask 3:

The digitally converted values of the ADC unit lie between 0 – 1024 (10bits).

Use the data from the light sensor and **invert the behavior of the functionality**. Change the function *"showData(value)"* so that 5 levels of light intensity are indicated, utilizing LED4. For minimum light incidence LED0 to LED4 should glow.

2.5.   Task 3

The PIT (Periodic Interrupt Timer) module on the board provides 4 different timer channels. In the following tasks digital inputs and timer-functionality will be used.

Subtask 1:

Similar to the LEDs, the buttons and switches *"BT5"*, *"BT6"*, *"SW1"* etc., are also mapped to each pin of the controller. The correct *"PCR"* register can be found by following the same procedure as LED configuration. A button can be configured as input by writing *"0x0100"* to the corresponding PCR register.

A button press or a switch on action results in a low signal (*"0"*) on the pin. This signal can be checked by the input register of the corresponding pin (*"SIU.GPDI[X].R"*, where X stands for the corresponding *"PCR"* register number).

Configure buttons BT5, and BT6 as input and indicate button presses by switching on LED5 and LED6.

Subtask 2:

The PIT timer channels 0 and 1 have already been configured. The timers can be started by calling the function *"PIT_StartTimer(int timerChannel)"*. To stop the timers the
"PIT_StopTimer(int timerChannel)" can be called.

Before a timer is started it must be configured. This can be done by calling *"PIT_ConfigureTimer(int timerChannel, unsigned int time)"*. The timer should be configured with a valid timer channel and a valid time. Thereby the time defines the period in milliseconds an interrupt service routine is executed. When the interrupt occurs, the function *"PITCHANNEL0()"* is being called. To indicate a working timer, **LED7 should be toggled at every second**. *"SIU.GPDO[X].R = ~SIU.GPDO[X].R"* (where X stands for the *"PCR"* register number) can be used to let the LED blink.

Consider the **"~"**-Operator, which negates the following value, in this case the current value of the register for the corresponding LED.

The implementation should be an endless loop. Thereby **the timer channel should be switched on/off using SW1**, if the switch state is changed. Implement this functionality.

This task shows a part of the indicator control of an automobile. In a later practical unit the indicator control will be expanded of the possibilities of left, right and hazard flashing.

2.6. Task 4 (Optional)

Use LED0 – LED7 to implement a "running light". Objective is to switch on the LEDs one by one starting with LED0. If all LEDs are on, they should be switched off successively starting with LED0 again.

List of References

[1] Schäuffele, Jörg; Zurawka, Thomas: *Automotive Software Engineering.* Wiesbaden: Springer Vieweg, 2013

[2] STMicroelectronics: *SPC560P50Lx Family Reference Manual.* URL: http://www.st.com/stweb-ui/static/active/en/resource/technical/document/reference_manual/CD00204027.pdf (23.03.2016)

[3] Raisonance S.A.S.: *REva v3 User Guide.* 2010